To:

Daniel A. Morales,

NJAS,

C. R. Greathouse IV,

and

T. D. Noe.

Re-define the Primes?... Why not?

R. J. Cano, <remy@ula.ve>

November 15, 2012

Abstract

Briefly, some ideas about the concept of prime number and its computer aid treatment will be described here.

Contents

1 An obvious starting point: The classic definition of prime number.

1.1 Definition:

A positive integer (sometimes also called a Natural number) is prime if it has no more divisors than itself and the unit.

1.2 The obvious algorithm: alg01

According to the previous definition and in order to determine whether a positive integer is prime or not, we pick each integer smaller than it and test by trial successive divisions. If some of the those smaller numbers divides the subject of the test, it is not prime, but if decreasing the divisor by one unit each time we reach the unit, then we may be sure that the subject of the described test is a prime.

The process here described might take too much time of computation (regardless making it manually by hand writing down everything in paper, of course we are referring here to the modern computers. In fact, incredible efforts are made continuously by the human kind in order to upgrade the knowledge about what is called number theory, included there the prime numbers), but we might pay enough attention to the operational structure of such algorithm realizing an interesting consequence of the "classic" definition for prime numbers...

2 The humble programmer's 1st idea:

2.1 Description:

Each time we find a prime number when executing "alg01", it will be clear that every smaller integer won't divide it. Thinking a little about such behavior, it is not difficult to conclude that a prime number doesn't divide the factorial built by all the integers preceding it, being this statement in some way an alternative definition reciprocal to the classic one in the sense of the rational fractions. Conceptually, such alternative definition would be related to the pentadic (5-adic) valuation of a integer number used for example to determine the number of trailing zeros in a factorial.

In other words, and in a closer description to a formal treatment, we state based on the observed property that, any given integer of the form $(K+1) \neq 4$ is prime if it satisfy:

$$\frac{k!}{(k+1)} \notin \mathbb{Z}$$

And here we must emphasize that what would be prime is $(k+1)$ instead of k. This alternative definition necessarily should be complemented with the explicit statement that the number 4 is not a prime. This definition brings a nice feature: It justify why the unit is not a prime number (please set k=0

and try it). Also it might be checked against the classic definition matching successfully for all the positive integers (Only when it has been included the complementary statement about the number $(k+1)=4$).

Moreover, this alternative definition is helpful to draw as logically true the important conjecture made by C. Goldbach in a letter sent by him to L. Euler; It is in good agreement with the apparently true fact that the sum of two primes gives an even number. However the uncertainty about if it is possible to find a counter-example of such conjecture remains unsolved.

2.2 Similar, but not the same algorithm: algo02

A second algorithm $algo02$ using the alternative definition for identifying primes, should be faster than the one based on successive divisions (algo01), and if we notice that it might be "filled" only with even numbers (those what might generate prime numbers through the described factorial criteria) the resulting execution should be faster: Now we are checking over "the half" of the integer subset studied by testing our alternative criteria only for even numbers, and apparently avoiding the successive divisions loop because assuming that the execution environment doesn't save extra information about the decomposition of the computed factorial then in each iteration it should make just a single division (just as if it doesn't know that the given numerators are factorials).

3 The humble programmer's 2nd idea:

3.1 Description of an experimental storage format for prime numbers.

Inspired by the simpleness of those computer programming/scripting languages like C or PARI, specifically by the absence of built-in Boolean constructions similar to the $true/false$ keywords and expression results like those implemented either in MAPLE or Pascal, a library or user defined $isprime\,()$ function should return only either 1 or 0 (both interpreted as true or false respectively). We might use such returned value as the toggle switch for the N-th bit inside an arbitrary precision unsigned integer context for code a finite integer mapping of the prime numbers in a single value of the form ($\lambda \geq 1$ is the higher integer for which the primality is already precomputed):

$$primesMagicNum\,(\lambda) = \sum_{offset=1}^{\lambda} \left\{ isprime\,(offset) \cdot 2^{offset} \right\}$$

And now by doing it so, we would be able not only for determining whether a given integer is prime or not faster than algo01 and algo02 just by using the binary AND, but also for finding the prime factorization (theoretically possible) of any given integer, checking first for those primes coded and stored in $primesMagicNum\,(\lambda)$ that divide the given number in order to know the correct set of valuations that would be necessary.

Better is this yet: After a divisor is known, by trial we might look for those exponents that vanish every prime initially valuated on the original number to factorize.

Finally[1], an alternative implementation of $isprime\,()$ based on the map stored as $primesMagicNum\,(\lambda)$ should look similar to[2]:

$$anotherIsPrime\,(x) = \quad primesMagicNum \quad \mathbb{AND} \quad 2^x$$

And here for instance, in C we might use the already optimized &= operator.

Another hint also is that the bit-shifting operations (in some languages called shl, shr and so on) might be used in replacement of a power operator like the double-asterisk or the circumflex accent for taking a power of 2, since such shifting operations are equivalent to multiplying or dividing by 2.

[1]The final drift and motivation for writting this draft raised in the middle of the work for completing the entry A218976 at OEIS.org after studying the modular equation:
$(\ x\ Mod\ y\ ) - x + y = 0$

[2]This time the λ enclosed between parenthesis is omitted in order to emphasize the idea of a "constant" number resting in the primary memory at the execution environment of every computer program applying this ideas (after such "constant" has been either computed or loaded just once and while the execution takes place).